Legacy Desktop Client Deployment Automation

Professional portfolio project summary

Amandio Pereira | UK Professional Standards Organisation | 2025

This case study explains how I used PowerShell and Intune to automate an eight-step manual installation, complete the remaining configuration after reboot, and let users uninstall the application through Company Portal.

Organisation-specific identifiers, application names and internal paths have been generalised for public presentation; the core technologies, workflow and engineering decisions are retained.

Project objective	Bring a critical legacy client into the cloud-managed endpoint model, replacing an eight-step manual installation with Company Portal delivery and managed removal.
My role	Senior technical lead who proposed and secured support for the endpoint modernisation, designed its Entra ID, Autopilot and Intune implementation, and developed, tested and documented this four-script PowerShell solution.
Business value	Users can now install the application through Company Portal without IT having to install it manually on each laptop. Deployment logs were added to help the team identify where a problem occurred and investigate its cause.
Core technologies	PowerShell 5.1, Intune Win32 apps, Company Portal, Windows Scheduled Tasks, registry automation, ServiceUI and WMI; delivered within the wider Entra ID and Autopilot solution.

1. Executive Summary

Towards the end of my one-year contract, a critical legacy desktop client remained the last work application requiring manual installation by IT. I had already established the Entra ID, Autopilot and Intune deployment approach, but this application was no longer supported and its bespoke design offered very limited deployment options. Installation involved eight steps requiring IT intervention before, during and after a reboot. At first glance, automation seemed unlikely. Nevertheless, I tackled this final obstacle as I usually would, on the basis that a working manual procedure could be understood and translated into code.

To understand the process properly, I worked through it with the most experienced member of the IT team, documenting the application's quirks, dependencies and the purpose behind each step. The vendor installer handled only the client installation itself. Three legacy runtime DLLs then had to be copied into SysWOW64, the Windows folder used for 32-bit system components on 64-bit Windows, and separate machine-level registry settings imported. After a reboot, the client needed an elevated first launch to initialise it, followed by user-specific registry settings applied to existing profiles.

With that sequence understood, I set out to overcome the application's deployment limitations by combining PowerShell scripting with Intune's built-in automation, making it available for users to install through Company Portal. This took me beyond my comfort zone into more involved programming. While AI helped me develop the scripts, I still needed to understand how and why each part worked, test its behaviour and decide whether it met the deployment requirements. Key design decisions were only made possible by examining the installation process in detail.

Completing the deployment and removal workflow

I divided the eight manual steps into stages that PowerShell and Intune could carry out in sequence. The first script runs in SYSTEM context, checks for the installation shortcut and applies the machine configuration. It also copies the files needed after reboot onto the laptop and creates a scheduled task to continue the work at the next logon.

After reboot, that task starts the second script with the elevated rights needed for the client's first launch. The script then targets the user registry settings at existing profiles, adds the final desktop shortcut and removes the temporary deployment files.

2. Starting Point and Engineering Constraints

The starting point was a working but technician-dependent eight-step installation process. Mapping the procedure showed that automation had to preserve its sequence, distinguish machine configuration from user settings, and continue after a reboot. The table connects each manual action with the requirement it placed on the managed deployment.

Original Manual Deployment Process

Step	Manual action	Requirement for automation
1	Run the vendor installer with administrative rights.	Provide the required rights through Intune's SYSTEM execution.
2	Wait for the client to appear and confirm installation.	Check for the expected shortcut before starting configuration.
3	Copy the additional configuration content onto the device.	Keep the continuation script and user configuration available locally after reboot.
4	Copy three runtime DLLs into SysWOW64 and import machine registry settings.	Apply dependencies and machine configuration that the vendor installer did not supply.
5	Reboot before continuing.	Stage the files and register the logon task before the managed restart.
6	Launch the desktop client once with elevated rights.	Reproduce the initialisation step through the SYSTEM scheduled task.
7	Apply the required current-user registry settings.	Direct the registry template towards existing user profiles rather than SYSTEM's own user hive.
8	Open the application for use.	Complete initialisation and configuration, then publish the final public desktop shortcut.

Uninstallation added a separate constraint: its prompts required a visible desktop session. Administrative rights alone would not make a background prompt accessible to the signed-in user.

3. Solution Design

I translated the eight manual steps into four scripts with separate responsibilities: initial installation, post-reboot completion, preparation of the interactive uninstall, and final removal. The scripts and their supporting files were packaged as one Intune Win32 application. This kept the application lifecycle within the endpoint-management approach I had introduced.

The script names below are descriptive public aliases. The package included the vendor installer, two registry files, three runtime DLLs and ServiceUI, with continuation files staged locally before reboot.

3.1 Install.ps1 — Initial installation and orchestration

Purpose	Install the client, apply the additional machine configuration and prepare the continuation after reboot.
Key inputs	Vendor installer, machine registry file, user registry template, runtime DLLs and the post-reboot script.
Logic	The script creates the log folder if needed and records the start of installation. It checks whether PowerShell is running in 64-bit mode and includes a SysNative relaunch path for 32-bit execution. It then resolves the package location, prepares the temporary working folder and starts the vendor installer silently. After the installer finishes running, the script waits 20 seconds, then checks for the application shortcut every five seconds for up to eight minutes. It only continues with configuration if the shortcut is found. It then copies the user registry template and post-reboot script onto the laptop, places the three DLLs in SysWOW64 and imports the machine registry file. A scheduled task is registered to run the staged script as SYSTEM at logon, including when the laptop is on battery power. This prepares the handoff before the restart handled by the Intune deployment configuration. The source script's direct Restart-Computer command is commented out; the script itself prepares for the reboot rather than forcing it.
Main output	The client has reached the installation checkpoint, its additional machine settings have been applied, and the files and scheduled task are ready to continue configuration after reboot.

3.2 After_Reboot.ps1 — Initialisation and user configuration

Purpose	Carry out the elevated first-launch stage and user configuration, then expose the final shortcut and clear temporary deployment assets.
Key inputs	Installed client, staged user registry template, existing non-special user profiles and the continuation task.
Logic	The second script appends an AFTER REBOOT marker to the same installation log. It starts the client for initialisation, records its process ID and allows a short startup interval before the process-cleanup sequence. It then checks that the user registry template is present before continuing. For each existing non-special user profile, the script records the SID, replaces HKEY_CURRENT_USER in the template with HKEY_USERS\<SID>, and imports a temporary profile-specific registry file. This stage configured profiles already present on the device; it did not provision settings for profiles created later. The script targets existing user profiles from SYSTEM, avoiding a separate registry import by the user. The script copies the final shortcut to the Public Desktop, then removes the scheduled task, temporary registry file and obsolete vendor shortcuts. Finally, it arranges its own deletion. The installation log is kept so the team can review what happened before and after reboot.
Main output	The post-reboot sequence has run, the shortcut is published and temporary assets are cleared, with actions and explicitly handled failures recorded in the log.

3.3 Uninstall-Prepare.ps1 — Interactive uninstall handoff

Purpose	Make the uninstall prompts available in the user's desktop session while retaining the SYSTEM rights needed for removal.
Key inputs	Package location, ServiceUI.exe, the main uninstall script, the explorer.exe session and a temporary scheduled task.
Logic	The preparation script resolves and logs the package root, ServiceUI path and main uninstall script path. It then builds the ServiceUI command using -process:explorer.exe, allowing the next stage to display the vendor's prompts in the signed-in user's desktop session. A one-off scheduled task is registered to run as SYSTEM five seconds later. The script records the task name, start time and execution account, briefly waits for registration to complete, then verifies that the task exists before starting it. If registration or startup fails, the problem is logged and the script returns a failure exit code. This separates the delivery mechanism from the removal itself. If Company Portal successfully launches the preparation stage but no prompt appears, the recorded paths and task events help establish whether the failure occurred before the main uninstaller began. Intune provided no native mechanism for an interactive SYSTEM-context uninstall. ServiceUI was therefore used as a temporary bridge, exposing the vendor's required prompts in the signed-in user's session while retaining the elevated rights needed for removal. ServiceUI was included in the deployment package and invoked for the uninstall operation.
Main output	The main uninstall script is launched through a managed task and ServiceUI, with its prompts visible to the signed-in user. Preparation and removal append to the same uninstall log.

3.4 Uninstall.ps1 — Vendor removal and residual cleanup

Purpose	Remove the client and the extra files and settings that were supplied outside the vendor installer.
Key inputs	Vendor uninstall executable, application shortcut, copied runtime DLLs, application registry locations, public shortcut and uninstall task.
Logic	The removal script adds its own stage marker to the uninstall log and checks the PowerShell architecture. A topmost warning asks the user to defer the vendor's immediate restart so that the remaining cleanup can finish. The vendor uninstaller then runs in the visible session. After the uninstall process returns, the script waits for the application shortcut to disappear before continuing. It removes the three copied DLLs, processes application keys in the available user hives, and clears the machine-side registry configuration added for deployment. Registry cleanup removes a predefined set of keys and selected values associated with the application's configuration. The final steps remove the old installation log, public desktop shortcut and temporary uninstall task. The uninstall log records removal actions, items that are already absent and errors caught during task cleanup.
Main output	The vendor removal and additional cleanup sequence has completed, with a retained uninstall log for support review.

Managing deployment handoff

The reboot and interactive-desktop requirements are handled at different points for different reasons. Installation needs local files and a logon task because execution must continue after restart. Uninstall needs ServiceUI because a prompt launched under SYSTEM would otherwise remain outside the user's visible desktop. Separating those responsibilities made each path easier to investigate and maintain.

Later development: This case study records the solution as implemented in March 2025. PSAppDeployToolkit 4.1, released on 7 August that year, subsequently introduced a client/server UI model that allows SYSTEM-context deployments to present user interaction without ServiceUI. PSAppDeployToolkit 4.1 release announcement.

4. Selected Implementation Patterns

The following scripting examples are shortened and sanitised. They illustrate how I controlled installation progress, applied settings to user profiles and made the scripts easier to troubleshoot. The deployment used PowerShell 5.1 with explicit 64-bit execution to avoid unintended file-system and registry redirection.

Timestamped logging across separate executions

A small Write-Log function gives each event the same timestamp format and appends it to a known file. The installation and post-reboot scripts share the installation log, so reboot does not separate their history. The two removal scripts similarly share an uninstall log. Stage banners identify where each execution begins.

```
# The installation stage creates the log folder if needed.
$logFilePath = 'C:\ProgramData\DeploymentLogs\Client-install.log'

function Write-Log {
   param ([string]$Message)
   $timestamp = Get-Date -Format 'dd-MM-yyyy HH:mm:ss'
   Add-Content -Path $logFilePath -Value "$timestamp - $Message"
}

Write-Log '**** AFTER REBOOT LOG ****'
$process = Start-Process -FilePath $programPath `
   -WindowStyle Hidden -PassThru
Write-Log "Program started: $programPath (PID: $($process.Id))"
```

The path above is generalised. The function and process-ID logging follow the original install-side pattern. In the uninstall scripts, Write-Log also sends the same message to Write-Output, making it available through the script's output stream as well as the file.

Recording the information needed to investigate a failure

Recorded detail	What it helps establish
Stage markers and timestamps	Whether execution reached installation, continuation, uninstall preparation or removal, and in what order.
PowerShell architecture	Whether execution was in the expected Windows context when investigating registry or file redirection.
Package paths and ServiceUI arguments	Which files and command line the managed deployment actually used.
Client process ID and user SID	Which process was launched and which user profile the configuration loop was processing.
Task registration and startup	Whether the scheduled handoff was created, found and started.
Missing files, absent keys and caught errors	Whether a prerequisite was missing, an item was already absent or a handled operation failed.

Practical use of the logs

For example, a missing AFTER REBOOT marker would direct investigation towards the restart and task handoff. A missing registry template would point towards staging or cleanup. For an absent uninstall prompt, the preparation log would show the resolved paths, command line and task events.

The logs are kept for troubleshooting the current installation or its removal. A new installation deletes the previous uninstall log, and uninstalling deletes the installation log.

State-based installation checks with a defined timeout

The shortcut check controls whether the script proceeds to configuration. The example includes both the bounded wait and the branch that follows it, showing how an observed installation state determines the next action.

```
# Shortened from the installation script; names are generalised.
$maxWaitTime = 480
$elapsedTime = 0

while (!(Test-Path $shortcutPath) -and
      ($elapsedTime -lt $maxWaitTime)) {
   Start-Sleep -Seconds 5
   $elapsedTime += 5
}

if (Test-Path $shortcutPath) {
   Write-Log "Install checkpoint reached: $shortcutPath"
   # The original script stages files, applies machine
   # configuration and registers the continuation task here.
} else {
   Write-Log "Installation checkpoint not reached: $shortcutPath"
   # Additional configuration is skipped in this branch.
}
```

After the installer finishes, the script checks every five seconds for the application shortcut, allowing up to eight minutes (480 seconds) for it to appear. If found, configuration continues. If it is still missing, the script logs the problem and skips the remaining configuration steps.

Translating user settings while running as SYSTEM

The registry template addresses the difference between the account running the script and the user who needs the settings. HKEY_CURRENT_USER refers to the account executing the import. Under SYSTEM, the template therefore needs to target the intended user's HKEY_USERS path explicitly.

```
# Extract from inside the existing profile loop.
# $sid identifies the target profile; its hive must be available.
Write-Log "Importing registry settings for SID: $sid"
$tempReg = Join-Path $stagingPath "UserSettings-$sid.reg"

(Get-Content $userReg) -replace `
   'HKEY_CURRENT_USER', "HKEY_USERS\$sid" |
   Set-Content -Path $tempReg -Encoding ASCII

Start-Process -FilePath 'reg.exe' `
   -ArgumentList "import `"$tempReg`"" -Wait -NoNewWindow
Remove-Item -Path $tempReg -Force -ErrorAction SilentlyContinue
```

This excerpt shows the template conversion and per-profile temporary file, which were the relevant ideas in the original code. It deliberately leaves hive discovery and loading outside the example. The supplied implementation does not inspect the registry process exit code, so the log's completion message should be understood as the loop finishing rather than proof of a successful import into every profile. During testing, I confirmed the required registry values and application behaviour in user profiles covered by the deployment.

Making errors useful during troubleshooting

The uninstall task cleanup combines try/catch with ErrorAction Stop and logs the task name together with the caught error. This makes a failed cleanup action easier to identify than a generic final completion message, while keeping the diagnostic pattern small enough to reuse.

5. Maintainability and Operational Design

I kept the scripts divided into clear stages so colleagues could follow the sequence and investigate problems. Comments link the code to the original manual steps, explaining why each action is needed. Logs record progress, and cleanup removes temporary files and tasks once their work is complete.

Design choice	Implementation	Operational value
Separate responsibilities	Four scripts cover installation, continuation, interactive preparation and removal.	A support investigation can follow the relevant stage and its handoff.
Local staging	The continuation script and user registry template are copied onto the endpoint.	Post-reboot execution does not depend on the original package path remaining active.
Explicit cleanup	Temporary tasks, staged configuration and obsolete shortcuts are removed after their role ends.	Reduces deployment residue and leaves fewer temporary components to support.
Documented intent	Manual-step references, dates and explanatory comments accompany the implementation.	Colleagues can understand why a step exists before changing it.

6. Outcome and Value Delivered

Users can now install the client through Company Portal, including the additional configuration needed after reboot. They can also start its removal from the same place and respond to the vendor's prompts on their own desktop. IT no longer needs to carry out the eight-step installation on each laptop, making the application easier to provide to staff working remotely.

Outcome	Delivered change	Practical value
Remote deployment	The legacy client was packaged for Intune and Company Portal within the Entra ID and Autopilot model.	Reduced routine installation work for IT and supported devices provisioned away from the office.
Staged completion	Local configuration files and a SYSTEM logon task carried work beyond the initial execution.	Reproduced the application's unusual reboot and elevated initialisation requirements.
Managed uninstall	A scheduled task and ServiceUI exposed the vendor's prompts before residual cleanup.	Made removal usable through Company Portal despite the interactive vendor routine.
Knowledge transfer	The procedure, implementation stages and logging approach were documented.	Made maintenance less dependent on remembering the original manual sequence or consulting its author.

Engineering approach

Understanding the existing procedure with an experienced colleague allowed me to preserve the application's dependencies and design the automation around them. That combination of technical direction, hands-on implementation and knowledge transfer also shaped the wider modernisation I led.

End of case study